Probabilistic Networks Basis Criteria of Quality Assurance

Anton BYKAU
Belarusian State University of Informatics and Radioelectronics, Minsk, Belarus

Abstract. The paper considers the application interface testing technology on the basis of probabilistic networks; it offers the criteria of reliability dependence measure for the different application modules sharing the same functionality; it proposes an expand criterion of the testing completeness by the measure of uncertainty of the modules reliability; it is shown how to use automated mechanism of test coverage estimation by using graphical elements general templates; it offers the technology of atomic test union into common model; it proposes a bug reporting mechanism, which allows to estimate the quality of application under test and the quality of automated testing itself.

Keywords. Automated testing, probabilistic networks, criteria of quality assurance

Introduction

The concept of software quality assurance is usually described as a system of attributes or characteristics that can be evaluated using appropriate metrics [1]. These metrics help evaluate the presence of the corresponding characteristic or attribute of quality in software. The main criteria for the quality of the software are listed in ISO 9126.

In this paper, two factors are considered. According to ISO 9126 standard they are: functionality as the ability of software to solve demanded by a user problem, under certain conditions, and reliability as the ability of software to support specific performance under specified conditions [2]. These factors are directly related to software testing.

According to Myers' a test is an execution of a program in order to find the differences between the current state of software and the required (as the evidenced of the error presence) [3]. Also, testing is a process of software evaluation, the assessment of its qualities according to some metrics [4]. The source of information for a testing process could be a source code, a structure of input data, a set of requirements and the model of a developed application. The terms "white box testing" and "black box testing" refer to whether the test is performed with a source code or with interface, both user interface and application programming interface provided by a module under test.

For each test strategy the appropriate criteria for the test coverage could be provided. Test coverage criterion is a metric for completeness of a testing assessment. It measures the variety of situations classes that are taken for testing. The greater the level of test coverage, the greater situations classes are covered (greater verity of situations are taken for a test), the more bugs can be detected.

The most common criteria for a black box are:

- The testing of main application functions.
- Required specification testing
- The testing of classes of input data
- The testing of classes of output data

For the last two criteria there are the following aspects of testing based on the nature of the data used for testing:

- The testing of a tolerance range
- The testing of length of the data set
- The testing of the order of the dataset.

The criteria for white-box testing are based on the source code availability of control flow directly during the execution of the code. The testing is performed according to the logic of the program and the reasonableness of its execution on the basis of the white box criteria. These criteria are:

- Function coverage
- Statement coverage
- Branch coverage
- Decision coverage
- Modified condition coverage

1. Quality assurance criteria modifications

In this paper, it is proposed to use some innovations in the process of automating the functional testing of software through interface. The test automation is performed using both a tester action recorder system and an interface-analyzing algorithm. It is proposed to analyze the composition of GUI elements for each state of the application under test. The analysis is performed using general UI elements patterns. It allows controlling the test coverage of all graphical elements and functions of the program.

It is proposed to use not only black-box testing criteria e.g. functions testing, input and output data classes testing but also white-box testing criteria. The interface state model allows testing in similar way as the white-box testing criteria e.g. combinatorial conditions coverage in interface terms.

It is proposed to classify the graphic elements of modules that share common functionality in order to use common tests. It allows associating the measure of reliability of the modules that share common functionality. As an outcome the assessment of testing completeness could be produced. This mechanism allows evaluating the priorities of running different automated tests. It gives the opportunity to adjust the testing plan (as a milestone of problem solving) by estimating the modules that either are not yet tested either. In other words, the priority of running tests could be estimated.

2. Some problems of testing Web applications

The process of large software systems development usually faces requirements change. As a result, developers are adjusting up to 80% (an average) of application code [5],

and automated tests ought to be updated too. The process of maintaining automated tests up to date is one of the most time-consuming tasks of automation [6]. This happens because the automated tests are not integrated or the test code combined incorrectly. Due to the fact that most of application code is combined in libraries, force test developers also combine automated test code into common library. This library is developed specifically for a current project. Each of such libraries has a similar structure, however, the usage of one library on different projects is impossible.

There are some differences between the automation GUI testing process and the application developing. An automated tests developer has an object for tests writing - the application under test. The automated tests developers have to adjust their tests to new application interface versions. This is the cause of a future work planning problem and a bad design of automated tests common functions. The writing of a convenient library is a very complex task. It requires a good further development plans description. Unfortunately, such information is often misses. Also, as a result of a large number of separated updates to the general methods, the common functions code quality becomes worse and worse.

Another problem of GUI test automation is UI elements test coverage estimation. The GUI elements of program under test can be removed and added during a development process. Unfortunately, there are no tools for monitoring such sort of changes. Also, there are no tools for estimation of Web UI tests coverage.

Most projects use Unit testing as a means of GUI test automation, however, for such kind of testing it is necessary to perform a lot of preparation steps. It makes a developer to design dependent tests or included preparation steps before each test. As a result, some steps are performed repeatedly a grate amount times and the result of their work is scattered in various test results.

Another disadvantage of Unit testing is bugs reproduce relation, but it's difficult to relate bug tracking system with an automated GUI test tool. Most of the Unit test frameworks don't allow binding specific drops of tests with a corresponding defect. Falling tests have to be manually excluded from the testing process due to bug fixing.

1. Probabilistic networks basis technology of Web application interface test automation

The using of Selenium test tools for GUI test automation showed that the most important part of the automated test system is a flexible language for defining graphical interface elements. This problem is well solved by using the XPath language. It allows creating flexible templates to find the graphic elements in the DOM of web interface.

A developer should abstract from a concrete UI element on the page and use generic UI element templates and common tests to simplify the automated tests maintenance. UI templates creation is a complex process, but it greatly simplifies the further support of automated test. Also, GUI tests generalization requires test data integration. The generalization of the test data allows applying the same data set for each class of the GUI elements according to their environment.

The generalization of tests using for different modules of application leads to the generalization of test results. The process of the generalization of testing reflects that the same methods of calculation and the processing of the input data are used for different parts of the interface. It looks like the same functionality testing in different environments. If the system receives different test results using the same input data

during the testing of various parts of interface that use the same module for data processing, in the results may be a probability describing correctness of module work for all parts of interface. Such probability can be used to predict the same tests fails for other parts of application.

Generalization testing also allows accelerating and reducing the monotony of the automation process. A tester may use a visual code generator classifying UI elements and application state. He could apply the existing element classes of new elements, the same adapting the common tests for the specific elements of the program. A tester could use ready-made data sets to perform a testing process according to the criterion of combinatorial conditions testing.

The code generator algorithm can be represented as a phased process and consists of both the interface analyze phase and the phase of a tester choice a test available for current application state. After the page is loaded test system applies the list of patterns according to the hierarchy of possible nesting graphic elements and creates an interface map. The result is the loading into the hierarchically nested array of rectangular areas and their classes. Finally, the test system creates a context menu designed to select ready-made tests for each UI element class. As a result, there should be enough for a tester to wait the end of the interface analysis procedure, call the context menu of the desired element and evaluate the correctness of the testing. This action menu can be changed by changing appropriate tests and data for UI elements class.

Unit testing automation is useful for white-box testing, but Unit test using as a means of GUI test automation is difficult for maintains. This problem can be solved by using application interface state model. The nodes of such kind of model is the key state of application interface and communications are the set of available operations. Interface model using allows organizing the testing of any available sequence of operations. The model acts as an interpreted description and the kernel of test system appears as interpreter.

Also, the model using allows verifying the testing process in response to the obtained results. The core of the testing system can find alternative ways of preparation steps, in the case of bug in a way of navigation, using a heuristic algorithm to search for alternative paths in the graph [7]. Also it allows combine tests, reducing the total time of testing. It leads to the goals based on the automated test system. The system could automatically find and execute the necessary preparation steps according to any test aim. This leads to a server base test system, which can provide the intermediate bug reports and results of testing, continue further testing in parallel. Such system allows updating the application interface model and testing new items, according to a new description without stopping the testing other modules.

Also application interface state model using can solve the bug tracking and automated test system integration problem. The number of states for defect life cycle should be increased up to four. It allows separating the automated tests correctness estimation from software quality estimation. This problem is well described by J. Myers [8].

If a tester found a previously unknown defect, he adds the defect into bug tracking system, adds a negative test or associate defect with the existing test. If related bug is opened and the test fails, the system shows the result in green for tester's team member and in red for application developers. This means that automated test working is proper and the application under test contains a bug. This way, tester implements self-testing functionality [9].

After the bug has been fixed the automated test system shows successfully passed test in green for application developers and in red for tester's team member. This behavior means that the defect is no longer reproduced, and its status should be changed to "closed". If the defect appears again the test system will automatically determine the cause of fails.

The result of the technology of UI test automation is an interface states model. This model consists of atomic tests union so that the end state of the previous testing phase becomes the initial state for the next testing phase. If a test is used for multiple functions testing or for the different application states, this common test corresponds to the several appropriate connections or the model. This generalization of the testing reflects the code reusing during application development. For these tests the results of testing could be expanded by the measure of correctness of module work for all parts of interface. Also, each link of application state a model (test) related to measure the uncertainty of the testing results for the current version of application. Such probability can be considered as the conditional probability. The condition appears both already obtained results of other similar tests, and the current test result for the previous version, which was obtained by regression testing.

To formalize these kinds of relationships and the probability is proposed to use a probabilistic network. These networks are used as a tool for decision-making, in case of contradictory data. The probabilistic networks is a base technology represented in this paper use a modification algorithm of the R. D. Schechter [10] algorithm. The modified algorithm allows processing the network information even in the presence of features in the network like:

- The communications of probabilistic networks can be translated in a several directions.
- The communication of probabilistic networks can conflict with each other.

The result inside the node can take multiple values. The condition of the nodes is described by the probabilities that the result takes the appropriate value. The sum of all probabilities inside the node is equals to one. It indicates that the response can reliably takes only one value. The fact that the network connection may conflict with each other leads to considering mutual influence of the facts and produces an approximation of the solution.

To describe the algorithm give an example of the algorithm work for two simple networks. For simplicity the results, let use the relationship between only two nodes, and let conditional probabilities equal either 1 or 0. The probability of the target node can be calculated by using the Bayesian formula according to the conditional probability stored in the properly connection

$$P(A) = P(A|B) \times P(B) \tag{1}$$

However, how algorithm calculates probability of the target node if the communication conflict with each other? Let we consider the following example (Figure 1).

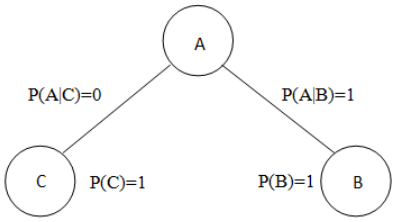

Figure 1. The contradictory evidence

In this example, the C-A and B-A communication can be considered as independent evidences, and the probability of A calculates as the probability of two independent events

$$P(A) = \frac{P(A|B) \times P(B) + P(A|C) \times P(C)}{2} \quad (2)$$

This formula can be applied to any number of connections converging into a single node. Another difficulty is the cycles in the network. Let's change the previously structure of the network connecting C and B. Assume that a given vertex A.

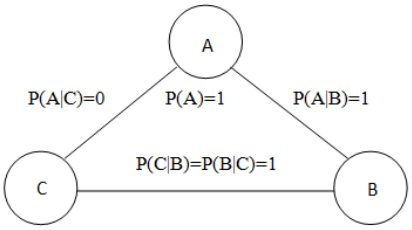

Figure 2. The contradictory relationships

For the network shown in Figure 2 is obvious the contradiction is the link from node A dictate the nodes B and C of the different states, but the connection C - B requires the identity of the values P(C) and P(B) of the nodes.

The decision of the controversy should be removing of one of connections out of the network, but before we do not know the relation to remove, we cannot do this. A temporary solution with equal confidence in the relations should be equal deviation of the node values B and C of dictated by the well-known characteristic P (A) and conditional probabilities - P (C = 1) = P (B = 0) = 0,333, P (C = 0) = P (B = 1) = 0.667.

This solution was obtained by excluding alternately connections out of the network, and averaging the results. Eliminate the link A - B: P (C) = 0, P (B) = 0, eliminate the link A - C: P (C) = 1, P (B) = 1, eliminate the link C - B: P (C) = 0, P (B) = 1.

The advantage of the algorithm is that the relationship can link more than two characteristics, and the algorithm inside the connection is limited by the requirement that for the same initial data algorithm should return the same result. Communication may be a function of several arguments in any programming language. The bidirectional communication between the two characteristics is described by the couple of connections oriented in opposite directions.

2. The main elements of the proposed Web applications testing technology

The technology allows organically combining manual testing and automation. The technology result is an integrated technology model. The model timely reflects the current charges of application under development.

The test system can automatically control the UI elements tests coverage and monitor all interface charges. The test system allows reusing of common tests for a several parts of the application interface with uses common functionality. The Interface states model using allows testing application interface according the criterion of combinatorial conditions testing of white box testing.

The use of probabilistic networks allows estimating the priority of the modules testing, using a measure of the uncertainty the testing results of individual modules, and the results for testing previous version of application.

The paper proposes to use more than two classical states of the life cycle of bug. The testing results are shown depending on the role of the person on the project. Although this innovation is aimed to integrate automated test and bug tracking systems, it allows evaluating the correctness of testing model and quality assurance the application under test separately. Also, this separation allows combining both the positive tests that verify the correct application behavior, and negative tests that verify the absence of incorrect behavior.

3. Criteria for application testing and test coverage estimation

In this paper it is proposed to use generalized patterns for each class of graphical components of Web application interface test automation. The use of generalized pattern of UI elements allows controlling the interface elements test coverage for all application states. It allows test application interface according to black-box criterion of functions testing.

The union of specific interface elements and their groups in the abstraction is performed by a tester as the reflection of functions and modules reuse for web application development. This criterion of test coverage is similar to the conditions combinatorial criterion for testing the white-box testing.

In this paper it is proposed to unite automated atomic tests into common model of application interface. The using application state model allows estimating the program functions reliability by testing application directly, and during the execution of preparatory steps.

The paper discusses the using conditional probability for estimation the connectivity of modules with the using common functionality. This measure can be used for calculation of uncertainty the results individual tests and it combinations according to obtaining the same tests result for other modules. The estimation of uncertainty tests results and tests combinations could be used for increase or decrease the test coverage for associated modules. Such measure could minimize the time of testing overwhelming majority of the functions application under test. It allows quickly verifying reproducing the maximum number of defects, minimizing the testing time. The test system is able to determine the priority of running the tests without human intervention, based on previously obtained results.

4. The main results

The paper discusses the probabilistic networks based on technology using application interface model. The mechanism UI elements test coverage of all application stats is described. The technology uses generalized patterns of UI elements to provide the testing according to black-box criterion of functions testing.

The paper describes the technology of atomic GUI tests combining into the common network. The nodes of the model are all the possible states of the application under test, and arches - all available operations on the interface. The using model of application interface states provides the testing according to criteria similar to the criterion of combinatorial coating conditions for white-box testing.

This technology is tested on the real world Ajax interface application testing, and has proven its effectiveness and convenience in comparison with the unit tests interface automation frameworks.

In this paper, a new measure for estimation the connectivity of test result of the different modules with using common functional is described. The Metrics of modules connectivity was suggested by Larry Constantine [4], which was also an early adherent of such concepts.

In this paper, the measure of priority of the testing the application modules for various combinations of conditions is proposed. The measure allows varying the test coverage of modules, obtained by analyzing the test results for the current or previous versions of the application.

The paper describes the mechanism of bug life cycle tracking, developed and tested by the author, allows evaluating the correctness of the automated tests, and the quality assessment of the application under test separately from each other. This separation is necessary because the system is automated testing and may contain bugs or may not correspond to the changed application requirements like any other program.

In this paper, a procedure that allows you to prepare the data to testing of tolerance range, testing of classes of input and output data is described.

Acknowledgement

The author thanks his scientific adviser I. Piletski for his help in preparing this paper.

References

[1] D. M. Marks, *Testing Very Big Systems*, Bellcore (McGraw-Hill), New-York, 1992.
[2] *ISO/IEC 9126-1, Software engineering – Product quality – Part 1: Quality model*, Geneva, Switzerland, ISO, 2001.
[3] G. J. Myers, *The Art of Software Testing*, Finance and Statistics, Moscow, 1982.
[4] V. V. Kulemin, *Methods of Software Verification*, Institute for System Programming, Moscow, 1995.
[5] I. Vinnichenko, *Automation of Testing Processes*, Peter Press, C-Petersburg, 2005.
[6] *Automated Testing Info* [Internet]. Available from: http://automated-testing.info/content/samoe-slaboe-zveno-vashey-avtomatizacii.
[7] S. Russell, P. Norvig, *Artificial Intelligence: a Modern Approach*, Williams, Moscow, 2007.
[8] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., New Jersey, 2004.
[9] J. Bicevskis, Z. Bicevska, and J. Borzovs, Regression testing of software system specifications and computer programs. In: *Proceedings of the 8th Software Quality Week*, San Francisco, paper 5-T-1, 1995.
[10] R. D. Shachter, Evaluating influence diagrams, *Operations Research* **34**(6) (1986), 871–882.